Extending GCM and Fscript for the Distributed Reconfiguration of Components

Boutheina Bennour, Ludovic Henrio, Marcela Rivera
INRIA – CNRS – Université de Nice Sophia-Antipolis
2004 Route de Lucioles 06902 Sophia Antipolis - France
{*bbenour,lhenrio,mrivera*}*@sophia.inria.fr*

I. Introduction

A. Context

This work is placed in the context of distributed computing, and especially large-scale distribution. Components has been considered over the recent years as a good abstraction to program distributed systems, thanks to the encapsulation they provide, their clearly defined interfaces The success of component models also comes from the high-level view of a component system (e.g. defined by an architecture description language) is considered as the good level to design distribution of applications.

Highly evolving distributed environments also require applications deployed on those environments to evolve. For this, some component models propose reconfiguration capabilities: the component composition can be reconfigured dynamically. This allows applications to adapt to changes in their execution environments, but also to changes in their functional and non-functional requirements.

B. Objective and Contribution

The objective of this paper is to increase the support for reconfiguration capacities in distributed component models. More precisely, we aim at allowing distributed systems to be reconfigured in a non-centralised manner. This article presents a distributed reconfiguration mechanism based on a scripting language.

To reach this goal, we suggest to extend existing component frameworks with two features:

- A controller, i.e. a non-functional port, localised in several (possibly all) components that is able to interpret reconfiguration orders.
- An extension of an existing scripting language for reconfiguration, adding primitives for distributed interpretation: remote execution of a reconfiguration script, and evaluation of script expressions to improve the passing of parameters between different execution contexts.

We show the adequacy of our contribution by providing an implementation of those features in the context of the Grid component model (GCM), and of its reference implementation above the ProActive middleware. We adapted the FScript reconfiguration language, that was designed for the Fractal component model to distributed component systems.

The purpose of this work is to help the programmer write reconfiguration procedures by providing him an adapted language. The programmer of the adaptation code focuses on the operations to be triggered thanks to the use of a scripting language, he can design distributed procedures for adaptation thanks to our extensions to the language and the component model.

The approach is designed for the GCM distributed component model because this is a distributed extension of Fractal. The Fractal component model is particularly well suited for component adaptability, because it comes with high reconfiguration capabilities, and a scripting language for reconfiguration: Fscript.

However the approach presented here can be adapted to other component models. It is not tight to the ProActive middleware that has been used for prototyping and experimentation. Roughly, the approach only relies on the possibility to create a non-functional interface to the components that will receive reconfiguration scripts, and the scripting language relies on the possibility to introspect and modify the component architecture.

II. A Controller for Reconfiguration

In order to trigger decentralized reconfigurations on distributed components, we suggest to incorporate a reconfiguration script interpreter into the component. The interpreter associated with a given component can operate a distributed reconfiguration calling remote interpreters which belong to its sub-components. Therefore, several (possibly all) components are required to expose script interpretation capabilities. Externalizing such non-functional features is possible in the Fractal model thanks to the component membrane which is made up of controllers. Our goal is to add a controller for reconfiguration in the component membrane. This controller will provide the interface allowing the invocation of a script interpreter. In fact, Fractal specification defines basic controllers ensuring component reflective abilities (introspection and intercession). However, the reconfiguration controller improves component reconfigurability since it can interpret high level scripts. The programmer of adaptivity procedure can thus focus on a high level language than the straightforward invocation of basic reconfiguration primitives.
The reconfiguration controller brings out through a programming interface useful interpreter services to handle scripts for reconfiguration.

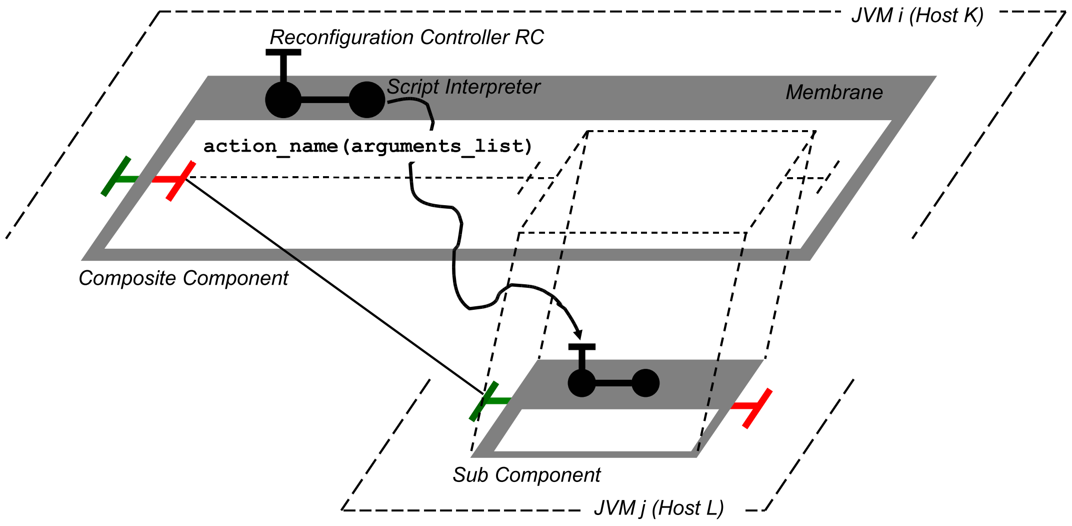

Fig. 1. Distributed interpretation of the reconfiguration action *action_name*

```
interface ReconfigurationController {
    void setInterpreter(String interpreterClassName)
    void loadScript(String scriptFileName)
    void executeAction(String actionName,
                            Object... arguments)
}
```

The method `setInterpreter` assigns an interpreter to the component. The method complies with the singleton pattern. If it does not exist, an interpreter instance is created. Actually, the interpreter is not instantiated when creating a component for performance efficiency: only components involved in a distributed reconfiguration process encapsulate interpreters.

Reconfiguration actions are defined in script files. The reconfiguration controller provides a method `loadScript` for the interpreter to recognize actions by parsing the script file.

Once the script file is loaded, a reconfiguration action can be triggered on the component by calling method `executeAction` of the controller API. The action name and arguments are passed as parameters to the method.

III. An Extension to the FScript language

A component, which has a reconfiguration controller, provides local control of reconfiguration since it is capable of performing its own script interpretation. Also, with the availability of a reconfiguration controller in its membrane, the component exposes interpretation features to its neighborhood. A script that reconfigures a composite component may be interpreted in part by a subcomponent or even by any other neighboring component. In order to delegate interpretation, we define a new primitive:

```
remote_call(target_component,
            action_name, arguments_list...)
```

The primitive `remote_call` triggers the execution of the reconfiguration action `action_name` by the interpreter associated with the component `target_component`. The first argument is an FPath expression that selects the node corresponding to the target component in the directed graph associated with the FScript interpreter. The second argument is a string that matches the reconfiguration action name. The arguments `arguments_list` of the action to interpret remotely are passed as parameters. The script programmer can thus specify actions to be performed on a remote component. Classically, arguments are evaluated locally, and then passed over to the remote script interpreter.

From this point in time, the target component becomes in charge of the interpretation of the reconfiguration. Unless the action is a primitive, the reconfiguration should be defined in the context of the target interpreter. To ensure the delegation mechanism, two pre-conditions are required:

Precondition 1 (Reconfigurability): The target component provides a controller for reconfiguration.

Precondition 2 (Global reconfiguration action): The target interpreter defines the reconfiguration primitive as a global action.

The remote call operation can only be applied to a component architecture if it satisfies the preconditions.

Figure 1 illustrates the remote interpretation of the reconfiguration action `action_name` by a target component which is here the subcomponent. Note that the components are deployed on different hosts, and therefore distributed: The composite component, respectively the subcomponent, is deployed on the host K, respectively on the host L.

IV. Conclusion

The originality of our approach is that it requires minimal extension to an existing component framework, and that it has been designed to provide specifically the key operations for the programmer of reconfiguration procedures.

We expect our work to impact the writing of adaptation procedures, and especially in the context of autonomic adaptation. Indeed autonomic adaptation of distributed components require components to self-adapt in a non-centralised manner. Without contributing to the design of self-adaptation procedures themselves, once these procedures will be designed, we expect this work to allow their fast and straightforward implementation.